ชนิดข้อมูลในภาษาซี

■ จเร เลิศสุดวิชัย

การเขียนโปรแกรมด้วยภาษาคอมพิวเตอร์ใด ๆ ก็ตาม ความเข้าใจในเรื่องของชนิดหรือแบบอย่างของข้อมูลและการใช้งานข้อมูล เป็นสิ่งสำคัญอันหนึ่งที่นักเขียนโปรแกรมไม่ควรจะละเลย ข้อผิดพลาดโปรแกรมบางครั้งก็มาจากการใช้ข้อมูลผิดลักษณะ โปรแกรมบางโปรแกรมอาจจะเขียนได้ง่ายและสั้นลงถ้าใช้ชนิดข้อมูลหรือออกแบบโครงสร้างข้อมูลที่ดี ดังนั้นในมุมภาษาซีฉบับนี้จะเป็นเรื่องราวของชนิดข้อมูลแบบต่าง ๆ ในภาษาซี ไม่ใช่เป็นการแนะนำ แต่จะกล่าวถึงรายละเอียดบางอย่างที่ควรจะรู้ในการใช้งาน

ข้อมูลแบบ Unsigned

ข้อมูลชนิด Unsigned char หรือ unsigned int ตามปกติจะเป็นข้อมูลแบบ char และ int ที่มีค่าเป็นบวกอย่างเดียวคือไม่สนใจเครื่องหมาย ซึ่งจะทำให้สามารถใช้ค่าตัวเลขมีช่วงบวกที่มากขึ้น แต่อย่างไรก็ตามข้อมูลแบบ unsigned ยังมีประโยชน์อีกอันหนึ่งที่ส่วนมากมักจะละเลยไป และอาจทำให้เกิดปัญหาในโปรแกรมได้

ตามปกตินั้นการเปลี่ยนชนิดข้อมูล (Type Convertion) จากชนิด char เป็น int (หรือ int เป็น long int) นั้น ถ้ากรณีที่ข้อมูลชนิด char (หรือ int) นั้นมีค่าเป็นลบ การเปลี่ยนชนิดเป็น int (หรือ long int) ค่าที่ถูกเปลี่ยนจะมีค่าเป็นลบเหมือนเดิมหรือไม่ คำตอบคือขึ้นอยู่กับเครื่อง-

```
#include <stdio.h>

main()
{
        char sc;
        unsigned uc;
        int si;
        unsigned int ui;
        long int li;

        sc = -128;      /* Hex value 0x80 */
        uc = 0x80;
        si = sc >> 4;
        printf( "The value of integer conver from char = %#x \n",si);
        si = uc >> 4;
        printf( "The value of integer conver from unsigned char = %#x \n",si);

        si = 0xe000;
        ui = 0xe000;
        li = si >> 6;
        printf( "The value of long integer conver from int = %#x \n",li);
        li = ui >> 6;
        printf( "The value of long integer conver from unsigned int = %#x \n",li);
}

The value of integer convert from char = 0xfff8
The value of integer convert from unsigned char = 0x8
The value of long integert conver from int = 0xfe00;
The value of long integert conver from unsigned int = 0x200
```

รูปที่ 1 โปรแกรมและผลลัพธ์แสดงข้อแตกต่างระหว่าง char และ unsigned

คอมพิวเตอร์ที่ใช้งาน (หรือสถาปัตยกรรมของคอมพิวเตอร์นั้น) ในบางเครื่อง char เมื่อถูกเปลี่ยนเป็น int ค่าที่เป็นลบเมื่อเปลี่ยนก็ยังคงเป็นลบอยู่ โดยการขยายบิตเครื่องหมายซ้ายสุดทั้งหมดและในบางเครื่องจะทำเพียงเติมบิตที่มีค่าเป็นศูนย์ไว้หน้าบิตเครื่องหมายจนเต็มเท่านั้น

สำหรับเครื่องที่เราใช้งานอยู่ คือ ระดับไมโครคอมพิวเตอร์จะเป็นลักษณะแรก เป็นแบบขยายบิตเครื่องหมายทางซ้ายจนเต็ม ตัวอย่างรูปที่ 1 ค่าในตัวแปร sc มีค่าเป็น -128 หรือในเลขฐานสองคือ 1000 0000 แต่เมื่อกำหนดค่าให้ตัวแปร i ค่าใน sc ก็จะมีค่าเป็น -128 เหมือนเดิม แต่ในเลขฐานสองจะมีค่าเป็น 1111 1111 1000 0000 ซึ่งใช้งานหรือโปรแกรมบางอย่างเราไม่ต้องการลักษณะนี้ แต่ต้องการในลักษณะที่สองที่กล่าวมาข้างต้น ซึ่งก็สามารถจะทำได้โดยเปลี่ยนชนิดของตัวจาก char(หรือ int) มาเป็น unsigned char (หรือ unsigned int) แทน ในรูปที่ 2 เป็นโปรแกรมแสดงข้อแตกต่างระหว่าง char กับ unsigned char เมื่อถูกเปลี่ยนเป็นชนิด int

ข้อมูลแบบโครงสร้าง Structure กับไฟล์ข้อมูล

ตามที่ทราบกันแล้วว่า structure เป็น

1	2	3	4	0	9	0	3	8	9	J	O	H	N		E		L	E	E	0	6	5	0	0	.	0	0

รูปที่ 2

```
#include <stdio.h>

struct date {
        int month;
        int day;
        int year;
};

struct {
        int pay_number;
        struct date pay_date;
        char name [11];
        float amount;
} employee;

main ()
{
        FILE *data;

        data = fopen ( "PAYMENT.DAT","rt");
        fscanf (data,"%4d%2d%2d%2d%10c%8f",
                &employee.pay_number,&employee.pay_date.month,
                &employee.pay_date.day,&employee.pay_date.year,
                &employee.name,&employee.amount);
        printf ("Pay number : %4d\n",employee.pay_number);
        printf ("pay date   : %02d/%02d/%02d\n",
                employee.pay_date.month,employee.pay_date.day,
                employee.pay_date.year);
        printf ("employee name : %s\n",employee.name);
        printf ("amount        : %8.2f\n",employee.amount);
        fclose (data);
}

Pay number  : 1234
pay date    : 09/03/89
employee name : John E lee
amount        : 6500.00
```

รูปที่ 3

การจัดเก็บตัวแปรในหน่วยความจำให้เป็นกลุ่ม ทำให้สะดวกการใช้งาน อย่างไรก็ตามบางครั้งอาจจะพบปัญหา เช่น ปัญหาเกี่ยวกับไฟล์ของข้อมูล ในการจัดการอ่านหรือเขียนไฟล์ข้อมูลเอง ตัวอย่างไฟล์ข้อมูลการจ่ายเงินเดือนพนักงานบริษัท ซึ่งลักษณะการจัดข้อมูลในไฟล์แต่ละเรคอร์ดเป็นดังตารางที่ 1 และสมมติว่ามีข้อมูลเป็นดังรูปที่ 2 และตัวอย่างการอ่านข้อมูลจากไฟล์ที่อยู่ในโปรแกรมรูปที่ 3

ซึ่งโปรแกรมในรูปที่ 4 เป็นโปรแกรมการอ่านข้อมูลจากไฟล์ที่มีโครงสร้างดังกล่าวมาข้างต้น ปัญหาที่เกิดคือ ในส่วนของฟิลด์ชื่อสกุลพนักงานจะพบว่ามีช่องว่างระหว่างชื่อและนามสกุล ถ้าโปรแกรมรูปที่ 3 เปลี่ยนการอ่านชื่อสกุลพนักงานเป็น %20s แทนฟังก์ชัน fsconf จะหยุดอ่านชื่อเมื่อพบช่องว่างช่องแรกคือจะอ่านได้แต่เพียงคำว่า "JOHN" เท่านั้น จากนั้นจึงเริ่มอ่านค่าของเงินเดือนที่จ่ายต่อไป ซึ่งค่าที่จะอ่านต่อไปจะเกิดผิดพลาด เพราะว่าตัวอักษร "E" ไม่สามารถเปลี่ยนเป็นค่า floating-point ได้

ตำแหน่ง	ฟิลด์
1 - 4	รหัสพนักงานบริษัท
5 - 6	เดือนที่จ่ายเงิน
7 - 8	วันที่จ่ายเงินเดือน
9 - 10	ปีที่จ่ายเงิน
11 - 21	ชื่อสกุลพนักงาน
22 - 29	จำนวนเงินที่จ่าย

ตารางที่ 1 ลักษณะการจัดข้อมูลในไฟล์แต่ละเรคอร์ด

การที่ใช้ฟอร์แมต %20c บอกถึงจำนวนตัวอักษรที่อ่านมี 20 ตัวอักษร โดยจะไม่ข้ามหรือหยุดเมื่อพบช่องว่าง ในกรณีที่ %s ในฟังก์ชัน fseanf (หรือ scanf) จะทำการต่อท้ายตัวอักษรด้วยรหัส '\∅' (บอกถึงการสิ้นสุด string) เองโดยอัตโนมัติ แต่ในกรณีของ %20c นั้น จะไม่มีการต่อท้ายด้วยรหัส '\∅' ซึ่งอาจพบปัญหาในการนำผลที่อ่านได้มาใช้งาน โดยจะปรากฏข้อความเกินเลยออกมา เกิดเนื่องมาจากไม่มีรหัสบอกการสิ้นสุด string ในกรณีนี้ผู้เขียนโปรแกรมจะต้องทำด้วยตัวเอง โดยเขียนโปรแกรมเพิ่มเติมบางส่วนลงไป

ข้อดีของการอ่านหรือเขียนไฟล์ด้วย fscanf หรือ fprintf นั้นคือสามารถจะกำหนดรูปแบบ(format)ของข้อมูลได้ แต่อย่างไรก็ตามการอ่านเขียนไฟล์โดยวิธีนี้จะช้า และข้อมูลบางชนิดจำเป็นต้องเก็บเป็นไบนารี่หรือเก็บทั้งไบนารี่และข้อความ ดังนั้นจึงเป็นหน้าที่ของผู้เขียนโปรแกรมที่จะต้องทำการเขียนฟังก์ชันสำหรับการอ่านเขียนข้อมูลในไฟล์เอง ซึ่งฟิลด์ที่เป็นชนิดไบนารี่สามารถจะอ่านได้ในลักษณะอะเรย์ของตัวอักษร แล้วทำการจัดเรียงใหม่อีกครั้ง (การสลับไบต์สูงกับไบต์ต่ำ เป็นต้น) เพื่อให้ตรงกับการจัดเรียงของโครงสร้างข้อมูล ทำให้การเขียนโปรแกรมยุ่งยาก

มีอีกวิธีหนึ่งสำหรับการอ่านหรือเขียนข้อมูลจากไฟล์ เก็บลงในข้อมูลชนิดโครงสร้างทั้งโครงสร้าง โดยไม่ต้องเสียเวลาอ่านทีละฟิลด์ ทำได้โดยใช้ฟังก์ชันในไลบรารีคือ fread และ fwrite ซึ่งจะต้องการกล่าวถึงการจัดเรียงหน่วยความจำของโครงสร้างข้อมูลเสียก่อน

การจัดเรียงหน่วยความจำของโครงสร้างข้อมูล (Structure Alignment)

ในภาษาซีไม่มีกฎอันใดตายตัวในการจัดเรียงหน่วยคำของข้อมูลแบบโครงสร้างขึ้นอยู่กับตัวคอมพายเลอร์และเครื่องคอมพิวเตอร์ ตัวแปรต่าง ๆ เช่นในเครื่อง DEC PDP-11 เริ่มต้นที่ตำแหน่ง word (ตำแหน่งคู่) long word (ตำแหน่งคู่ 4) ซึ่งก็จะทำให้เกิดช่องว่างของหน่วยความจำขึ้นในโครงสร้างข้อมูลที่บรรจุตัวแปรชนิดนั้นอยู่ด้วย อย่างไรก็ตาม ในเครื่องบางเครื่องก็ไม่มีความจำเป็นในเรื่องการเรียงตำแหน่งหน่วยความจำ เช่น เครื่อง PC XT ที่ใช้ไมโครโปรเซสเซอร์ 8086 แต่เพื่อเพิ่มประสิทธิภาพในการถ่ายเทข้อมูลก็อาจจะทำได้โดยการจัดเรียงหน่วยความจำลงในตำแหน่งคู่ได้

ในกรณีคอมพายเลอร์ภาษาซีบนเครื่องไมโครฯที่นิยมใช้งาน คือเทอร์โบซีไมโครซอฟต์ซี สามารถจะกำหนดการจัดเรียงหน่วยความจำ(Memory alignment)ได้เป็นทั้งแบบไบต์หรือเวิร์ด ลองพิจารณาดูโครงสร้างข้อมูล (Structure) ในรูปที่ 4 ในกรณีที่การจัดเรียงหน่วยความจำเป็นแบบไบต์

ตามในรูปที่ 1 โครงสร้างข้อมูลซึ่ง

```
/* struct - structure alignment example 1 */

struct st {
        char c;
        int i;
        char d;
};

struct st st1;
```

รูปที่ 4 ตัวอย่างโครงสร้างข้อมูลเรียงจัด alignment ของหน่วยความจำ

ประกอบด้วย ข้อมูลพื้นฐานชนิด char int และต่อท้าย char จะได้ขนาดของข้อมูลเป็น 4 ไบต์ แต่ถ้าเป็นการจัดเรียงหน่วยความจำแบบเวิร์ด (word) จะได้ขนาดข้อมูลเป็น 6 ไบต์ ซึ่งเป็นเพราะว่าข้อมูลในโครงสร้าง char ถึงแม้จะใช้พื้นที่หน่วยความจำเพียง 1 ไบต์ แต่การจัดเรียงข้อมูลตัวอื่นลงไปจะต้องลงที่ตำแหน่งหน่วยความจำเลขคู่ ซึ่งจะทำให้มองดูเหมือนกับค่าข้อมูลชนิด char ใช้หน่วยความจำถึง 2 ไบต์ ในกรณีโครงสร้างข้อมูลในรูปที่ 4 ถ้าทำการจัดตำแหน่งเสียใหม่ ดังในรูปที่ 5 โดยทำการสลับเอาข้อมูลชนิด char มาติดต่อ ซึ่งในลักษณะนี้ไม่ว่าจะการจัดเรียงหน่วยความจำแบบไบต์หรือเวิร์ดจะได้ขนาดของข้อมูลเท่ากันหมดคือ 4 ไบต์

```
            /* struct - structure alignment example 2 */

struct st {
        char c;
        char d;
        int i;
};

struct st st2;
```

รูปที่ 5 เปลี่ยนตำแหน่งองค์ประกอบในโครงสร้างข้อมูลของรูปที่ 1

```
struct date {
        int month;
        int day;
        int year;
};

struct {
        int pay_number;
        struct date pay_date;
        char name [11];
        float amount;
} employee;
```

รูปที่ 6

```
int fread (buffer, size, count ,stream);

void *buffer;           Storage location for data
int size;               Item size in byte
int count;              Maximum number of item to read
FILE *stream            Pointer to FILE structure

int fwrite (buffer, size, count ,stream);

void *buffer;           Pointer to data to be write
int size;               Item size in byte
int count;              Maximum number of item to write
FILE *stream            Pointer to FILE structure
```

รูปที่ 7

```
        fwrite (employee, sizeof (employee), 1, data);
        fread  (employee, sizeof (employee), 1, data);
```

รูปที่ 8 แสดงการใช้ฟังก์ชั่นการอ่านเขียนไฟล์

ครานี้ย้อนกลับไปพิจารณาปัญหาที่แล้วคือเรื่องของการอ่านเขียนไฟล์ในตัวอย่างที่กล่าวไปก่อนแล้ว โครงสร้างข้อมูลของโปรแกรมตัวอย่างที่ 1 ซึ่งจะยกมาเป็นรูปที่ 6 การอ่านเขียนไฟล์โดยใช้ฟังก์ชัน fread และ fwrite ซึ่งมีรูปแบบการใช้งานดังรูปที่ 7 การเขียนหรืออ่านไฟล์โดยฟังก์ชันนี้จะต้องกำหนดขนาดหรือจำนวนหน่วยความจำที่จะทำการอ่านหรือเขียน ตัวอย่างดังรูปที่ 8 ปัญหาที่เกิดขึ้นก็คือ ไฟล์ข้อมูลที่เกิดจากโปรแกรมที่ได้รับคอมพายล์โดยกำหนดการจัดเรียงหน่วยความจำไม่ตรงกันจะมีขนาดไฟล์ไม่เท่ากันหรือไม่สามารถใช้งานร่วมกันได้นั่นเอง นี่คือปัญหาอันหนึ่งที่บางครั้งอาจเกิดขึ้นมาแล้วมองข้ามไป ดังนั้นในลักษณะโปรแกรมที่ใช้การอ่านเขียนไฟล์แบบนี้ควรระมัดระวังเรื่องการจัดเรียงหน่วยความจำด้วย

ขอยกตัวอย่างอีกอันหนึ่งในเรื่องของการจัดเรียงหน่วยความจำเพื่อให้เข้าใจอย่างชัดเจน คือในตัวอย่างโปรแกรมในรูปที่ 9 มีตัวแปรที่เป็นโครงสร้างซึ่งภายในจัดวางตำแหน่งของส่วนย่อยแตกต่างกันออกไป ในกรณีที่เลือกการคอมพายล์โดยใช้การจัดเรียงหน่วยความจำเป็นเวิร์ด (ในเทอร์โบซีทำได้โดยเลือกเมนู Option, Compiler, Code generation, Alignment สำหรับไมโคร-

```
#include <stdio.h>

struct {
        char c1;
        int i1;
        char c2;
        char *pc;
        char c3;
        float f1;
        char c4;
} st1;

struct {
        float f1;
        char *pc;
        char c1;
        int i1;
        char c2;
        char c3;
        char c4;
} st2;

struct {
        char c1;
        char c2;
        char c3;
        char c4;
        int i1;
        char *pc;
        float f1;
} st3;

main ()
{
        printf ("Size of st1 = %d bytes.\n",sizeof (st1));
        printf ("Size of st2 = %d bytes.\n",sizeof (st2));
        printf ("Size of st3 = %d bytes.\n",sizeof (st3));


Size of st1 = 16 bytes.
Size of st2 = 14 bytes.
Size of st3 = 12 bytes.
```

รูปที่ 9 ตัวอย่างการจัดวางลำดับขององค์ประกอบในโครงสร้างข้อมูล

ซอฟต์ซี ใส่ option ใน command line -Zp) ผลของโปรแกรมจะเห็นว่าขนาดหน่วยความจำแต่ละโครงสร้างไม่เท่ากัน คือได้ 16, 12 และ 14 ไบต์ตามลำดับ ทั้งที่โครงสร้างภายในเหมือนกัน โครงสร้างแบบ st1 นั้นแย่ที่สุดเนื่องจากการแทรกตัวของข้อมูลแบบ char ระหว่างข้อมูลชนิดอื่น ซึ่งจะทำให้ถูกจัดเรียงโดยใช้ 1 เวิร์ดต่อ 1 char สำหรับโครงสร้างแบบ st2 นั้นรองลงมา ที่ดีที่สุดคือโครงสร้างแบบ st3 ซึ่งการวางตัวของส่วนย่อยภายในโครงสร้างสามารถจัด (pack) ให้ลงในตำแหน่งหน่วยความจำได้ดีที่สุด ดังนั้นจะเห็นได้ว่าการเรียงลำดับหรือตำแหน่งขององค์ประกอบในโครงสร้างข้อมูลมีผลต่อการจัดเรียงหน่วยความจำ ซึ่งถ้าการจัดลำดับหรือตำแหน่งถูกต้องก็จะทำให้การใช้หน่วยความจำมีประสิทธิภาพมากขึ้น

สำหรับมุมภาษาซีฉบับนี้คงจะทยอยลงเท่านี้ก่อน ในฉบับหน้าจะกล่าวถึงเรื่อง Bit field และ Union พบกันฉบับหน้าครับ

แถลงไข ซอฟต์เมล์ไมโครฯ

สวัสดีท่านผู้อ่านที่เคารพทุกท่าน หลังจากที่ทางวารสารได้จัดทำซอฟต์เมล์เพื่อความสะดวกสำหรับผู้อ่านนั้น ปรากฏว่ามีผู้อ่านหลายท่านได้สอบถามเข้ามาเกี่ยวกับการใช้ซอฟต์เมล์ดังกล่าว ดังนั้นเพื่อเป็นที่เข้าใจตรงกันเกี่ยวกับการใช้ซอฟต์เมล์ฉบับต่าง ๆ ของไมโครฯ จึงขอชี้แจงดังนี้คือ

- โปรแกรมหรือไฟล์ต่าง ๆ ที่ถูกบรรจุลงในแผ่นดิสเกตต์จะจัดแบ่งแยกตามบทความแต่ละเรื่องในฉบับนั้น ๆ โดยจะแบ่งแยกเป็นไดเรคทอรีย่อยของแต่ละเรื่องไป การย้ายเข้าไปในไดเรคทอรีใดให้พิมพ์ cd ตามด้วยชื่อไดเรคทอรีนั้น

- ก่อนที่ท่านผู้อ่านจะใช้โปรแกรมหรือไฟล์ดังกล่าว ควรจะอ่านบทความที่มีซอฟต์เมล์นั้นก่อน เพื่อที่จะได้ทราบว่าจะสามารถนำโปรแกรมหรือไฟล์นั้น ๆ ไปใช้ได้อย่างไร

ส่วนการ UNPACK ไฟล์ของซอฟต์เมล์ฉบับที่ 51 นั้น เฉพาะไฟล์ SARA.ARC นั้น แผ่นที่จะนำมาย้ายไฟล์นั้นควรฟอร์แมตแผ่นด้วยดอสเวอร์ชัน 3.1 หรือต่ำกว่า

หวังว่าท่านผู้อ่านคงจะสนุกกับการใช้ซอฟต์เมล์เหมือนเดิมนะครับ